CS 332 – Theory of Computation
Fall 2020

Lecture 1:

- Overview
- Course information
- Finite automata

Reading:
Sipser Ch. 0, 1.1

Course Information

Course Staff

- Me: **Ran Canetti**
  - Office hours: Thu 3:30-5, On Zoom (preferably)
  - Research interests: Cryptography and information security

- TF: Luowen Qian
  - Leading discussion sections
  - Office hours: Fri 2-4, On Zoon (prefereably)

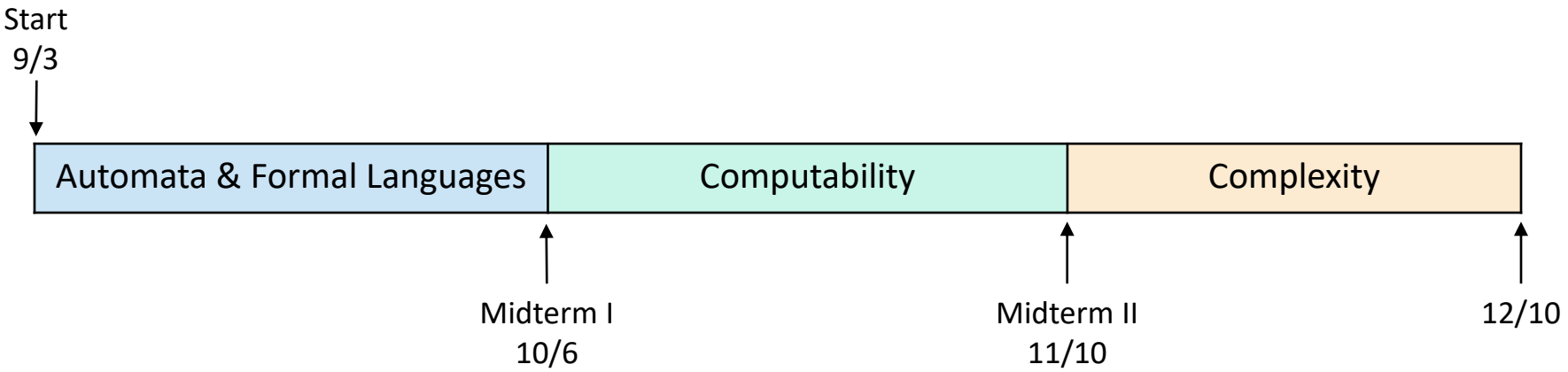
- Graders: TBD

Course Webpage

https://bu-cs332.github.io/

- Serves as the syllabus and schedule
- But the Piazza contains the uptodate information!

Course Structure (tentative)



Grading

- Homework (30%): Roughly 10 of these
- Exams (40%):
  - Midterm I (10%)
  - Midterm II (10%)
  - Final (20%)
- Participation (30%): TopHat, in class problems

9W7J53.

Homework Policies

- Weekly assignments due Tuesdays @ 2PM
- No late days, no extensions
- Lowest homework score will be dropped

- Homework to be submitted via Gradescope
  - Entry code: 9W7J53

- You are encouraged to typeset your solutions in LaTeX (resources available on course webpage)

- HW1 to be released on Tue 9/9, due Tue 9/16

Homework Policies: Collaboration

- You are encouraged to work with your classmates to discuss homework problems

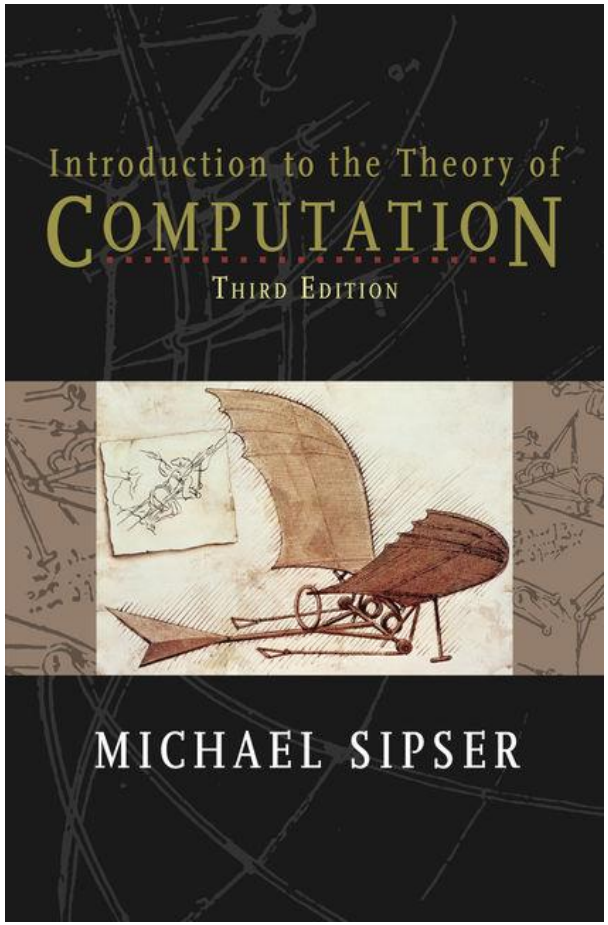
- HOWEVER:
  - You may collaborate with at most 3 other students
  - You must acknowledge your collaborators
  - You must write your solutions by yourself
  - You may not share written solutions
  - You may not search for solutions using the web or other outside resources
  - You may not receive help from anyone outside the course (including students from previous years or online helpers)

Homework Policies: Collaboration

Details of the collaboration policy may be found in Piazza

Important: **Sign** this document to affirm you understand it, and turn it in via Gradescope by 2PM, Tue 9/16

Textbook



Introduction to the Theory of Computation
(Third Edition)
by Michael Sipser

- It's fine if you want to use an older edition, but section numbers may not be the same
- Other resources available on course webpage

Piazza

- We will use Piazza for announcements and discussions
  - Ask questions here and help your classmates
  - Please use private messages / email sparingly

  https://piazza.com/bu/fall2020/cs332

  You can earn bonus points toward your participation grade by participating thoughtfully on Piazza

TopHat

- Your class participation score (30% of the course grade) will be determined by your engagement. We will be using TopHat:

https://tophat.com/ Entry code: 694168 or cs332f20

- Some of the questions will be graded based on participation only. Others will be scored on correctness.
- We will also be using TopHat for in-class problem solving in groups.

Expectations and Advice for Succeeding in CS 332

Our (the Course Staff's) Responsibilities

- Guide you through difficult parts of the material in lecture
- Encourage active participation in lectures / section
- Assign practice problems and homework that will give you a deep understanding of the material
- Give detailed (formative) feedback on assignments
- Be available outside of class (office hours, Piazza)
- Regularly solicit feedback to improve the course

Your Responsibilities

- Concepts in this course take some time to sink in. Keep at it, and be careful not to fall behind.
- Do the assigned reading on each topic before the corresponding lecture.
- Take advantage of office hours.
- Participate actively in lectures/sections and on Piazza.
- Allocate lots of time for the course: comparable to a project-based course, but spread more evenly.

Prerequisites

This class is fast-paced and assumes experience with mathematical reasoning and algorithmic thinking

You must have passed CS 330 – Intro to Algorithms

This means you should be comfortable with:

- Set theory
- Functions and relations
- Graphs
- Pigeonhole principle
- Propositional logic
- Asymptotic notation
- Graph algorithms (BFS, DFS)
- Dynamic programming
- NP-completeness

Contact Ran or Luowen if you have questions about your preparation for the course

Advice on Homework

- Start working on homework early! You can get started as soon as it's assigned.
- Spread your homework time over multiple days.
- You may work in groups (of up to 4 people), but its better to think about each problem for at least 30 minutes before your group meeting.

- To learn problem solving, you have to do it:
  - Try to think about how you would solve any presented problem before you read/hear the answer
  - Do exercises in the textbook in addition to assigned homework problems

Advice on Reading

- Not like reading a novel
- The goal is not to find out the answers, but to learn and understand the techniques
- Always try to predict what's coming next
- Always think about how you would approach a problem before reading the solution
- This applies to things that are not explicitly labeled as exercises or problems!

Course Overview

Objective

Understand the idea of *computation:*

Objective

Understand the idea of *computation:*

-What is computation? What isn't?

Objective

Understand the idea of *computation:*

-What is computation? What isn't?

-What are the important / interesting/ salient aspects?

What are the non-important ones?

Objective

Understand the idea of *computation:*

-What is computation? What isn't?

-What are the important / interesting/ salient aspects?
  What are the non-important ones?

-What can be computed? What cannot?

Objective

Understand the idea of *computation:*

-What is computation? What isn't?

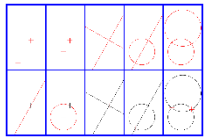
-What are the important / interesting/ salient aspects?
What are the non-important ones?

-What can be computed? What cannot?

➔ Build a *theory* of computation.

What is “computation”

- Examples:
  - Paper + pencil arithmetic
  - Abacus
  - Mechanical calculator
  - Slime mold
  - Ruler and compass geometry constructions
  - Java/C programs on a digital computer



- For us: Computation is the processing of information by the repeated application of a small set of *simple* and *local* operations.

What is a good "theory"?

- General ideas that apply to many different systems
- Expressed simply, abstractly, and precisely
- Captures the essence
- Enhances understanding

- Generality
  - Independence from Technology: Applies to the future as well as the present
  - Abstraction: Suppresses inessential details

- Precision: Can prove formal mathematical theorems
  - Positive results (what *can* be computed): correctness of algorithms and system designs
  - Negative results (what *cannot* be computed): proof that there is no algorithm to solve some problem in some setting (with certain cost)

Parts of a Theory of Computation

- Models for machines (computational devices)
- Models for the problems machines can be used to solve
- Theorems about what kinds of machines can solve what kinds of problems, and at what cost

This course: Sequential, single-processor computing

Not covered:

- Parallel machines
- Distributed systems
- Quantum computation
- Real-time systems
- Mobile computing
- Embedded systems

What is a Computational Problem?

A computational problem is represented by way of a function

$$f: D \rightarrow R$$

(D is the domain, R is the range).

A naïve representation of f : via a table

Elements of D	Corresponding value of f

Note: $D \rightarrow R$ can be infinite! (That's the interesting case…)

What is a Computational Problem?

We will concentrate on functions from *strings* to {0,1}.

(Or: recognizing whether a *string* is in a *language.*)

- **Alphabet**: A finite set Σ

  Ex. $\Sigma = \{a, b, \ldots, z\}$
- **String:** A finite concatenation of alphabet symbols (order matters)

  Ex. $bqr, ababb$

  The length of a string is the number of symbols.

  ε denotes empty string, length 0

  Σ^* = set of all finite strings over Σ
- **Language:** A (possibly infinite) set L of strings : $L \subseteq \Sigma^*$

Examples of Languages

Parity: Given a string consisting of a's and b's, does it contain an even number of a's?

Σ = L =

Primality: Given a natural number x (represented in binary), is x prime?

Σ = L =

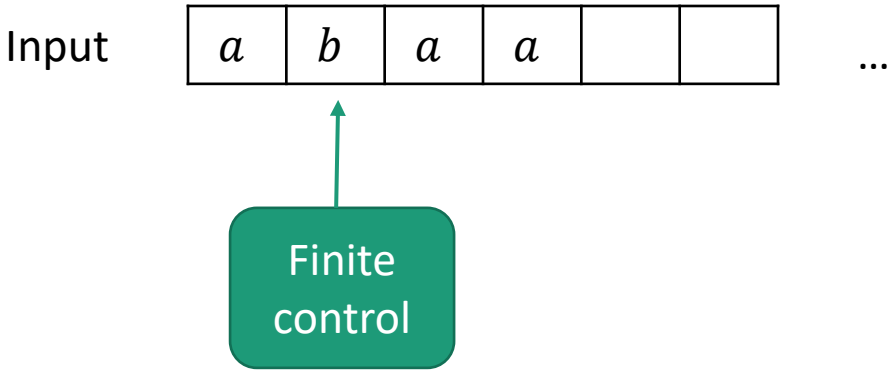
Halting Problem: Given a C program, can it ever get stuck in an infinite loop?

Σ = L =

Models of computation: Machines

Computation is the processing of information by the **repeated application** of a **small set** of **Simple** operations.

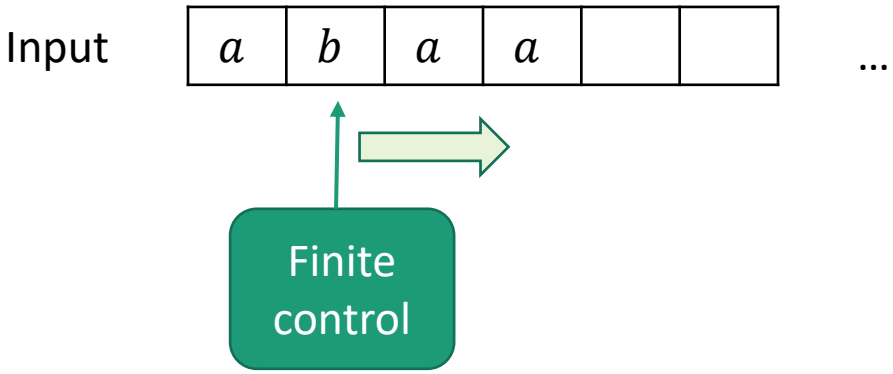
➔ What is the simplest "machine model" that can capture computation?



Abstraction: We don't care how the control is implemented. We just require it to have a finite number of states, and to transition between states using fixed rules.

Machine Models

- Finite Automata (FAs): Machine with a finite amount of unstructured memory

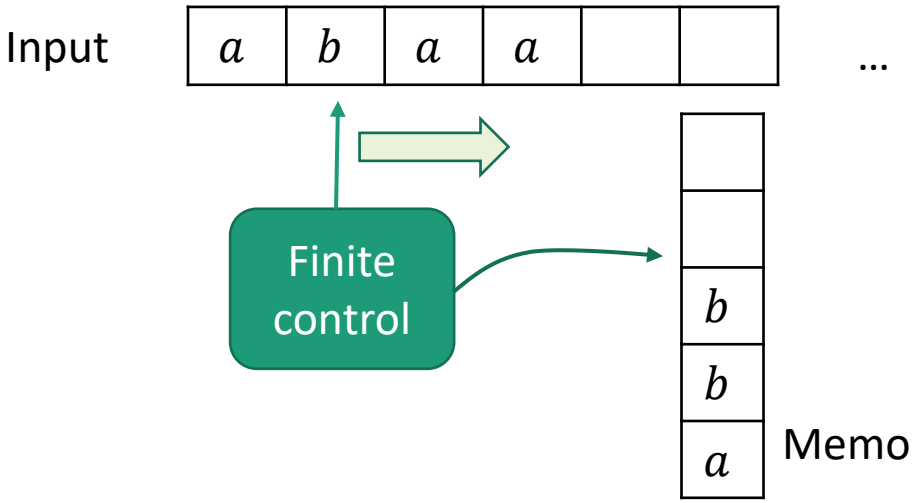


Control scans left-to-right
Can check simple patterns
Can't perform unlimited counting

Useful for modeling chips, simple control systems, choose-your-own adventure games…

Machine Models

- Pushdown Automata (PDAs): Machine with unbounded structured memory in the form of a stack

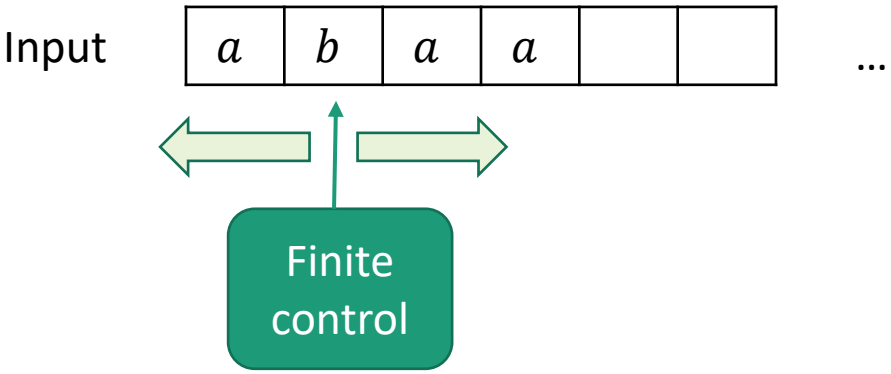


Control scans left-to-right
Can use stack to count, balance parentheses

Useful for modeling parsers, compilers, some math calculations

Machine Models

- Turing Machines (TMs): Machine with unbounded, unstructured memory



Control can scan in both directions
Control can both read and write

Model for general sequential computation
Church-Turing Thesis: Everything we intuitively think of as "computable" is computable by a Turing Machine

What theorems would we like to prove?

We will define classes of languages based on which machines can recognize them

Inclusion: Every language recognizable by a FA is also recognizable by a TM

Non-inclusion: There exist languages recognizable by TMs which are not recognizable by FAs

Completeness: Identify a “hardest” language in a class

Robustness: Alternative definitions of the same class

Ex. Languages recognizable by FAs = regular expressions

Why study theory of computation?

- You will learn how to formally reason about computation
- You will learn the technology-independent foundations of CS

Philosophically interesting questions:

- Are there well-defined problems which cannot be solved by computers?
- Can we always find the solution to a puzzle faster than trying all possibilities?
- Can we say what it means for one problem to be “harder” than another?

- This is the core of CS!

Why study theory of computation?

- You will learn how to formally reason about computation
- You will learn the technology-independent foundations of CS

Connections to other parts of science:

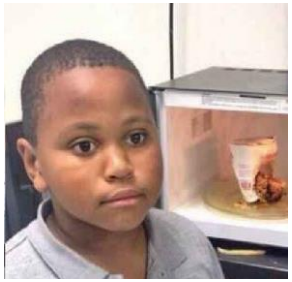
- Finite automata arise in compilers, AI, coding, chemistry

https://cstheory.stackexchange.com/a/14818

- Hard problems are essential to cryptography
- Computation occurs in cells/DNA, the brain, economic systems, physical systems, social networks, etc.

Why study theory of computation?

Practical knowledge for developers



"Boss, I can't find an efficient algorithm.
I guess I'm just too dumb."

"Boss, I can't find an efficient algorithm
because no such algorithm exists."

Will you be asked about this material on job interviews?

No promises, but a true story…